Introduction of DL

input: vector, Matrix

output: number(regression), type(classification), text, image

ML: look for the function

steps:

1. 写出一个带未知参数(W,b)的函数:

$$y = Wx + b$$

2. 定义Loss function

$$L = L(b, w)$$

3. 优化

$$w^*, b^* = argmin_{w,b}L$$

4.迭代，设定学习率σ

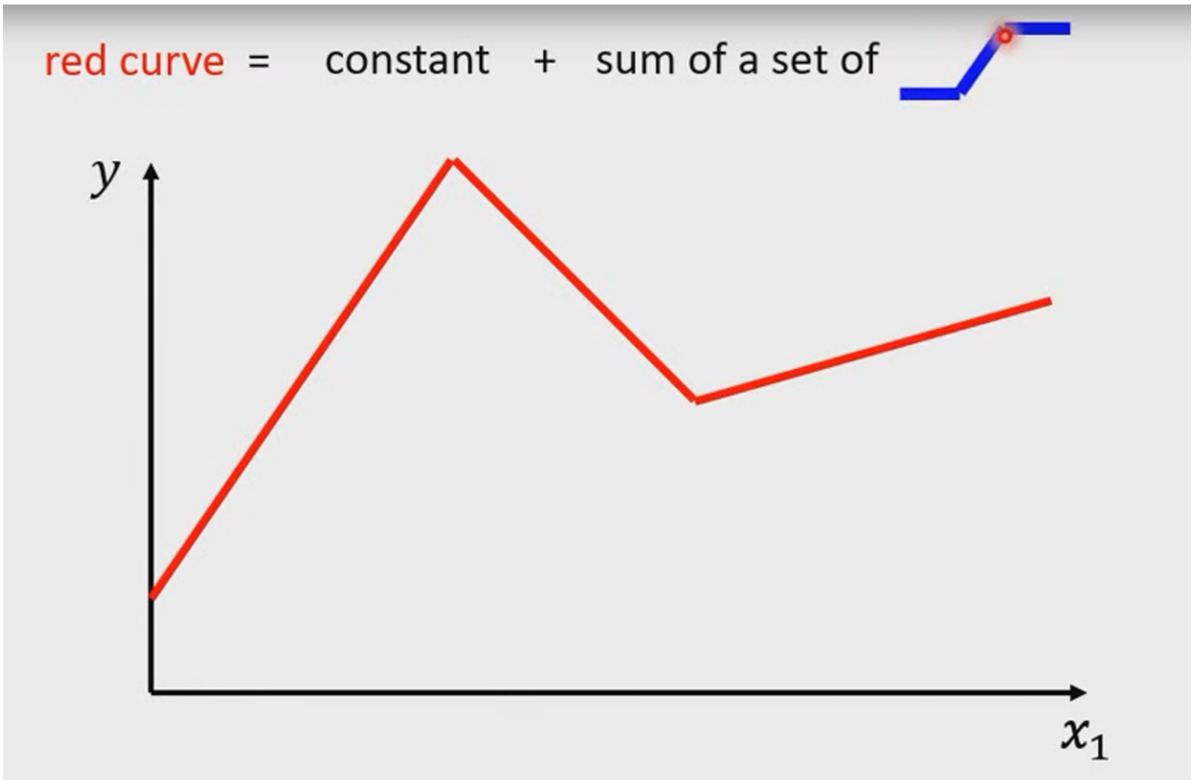
5. 修改，得到更好的模型

6. 也可以组合更多的参数

更好的模型

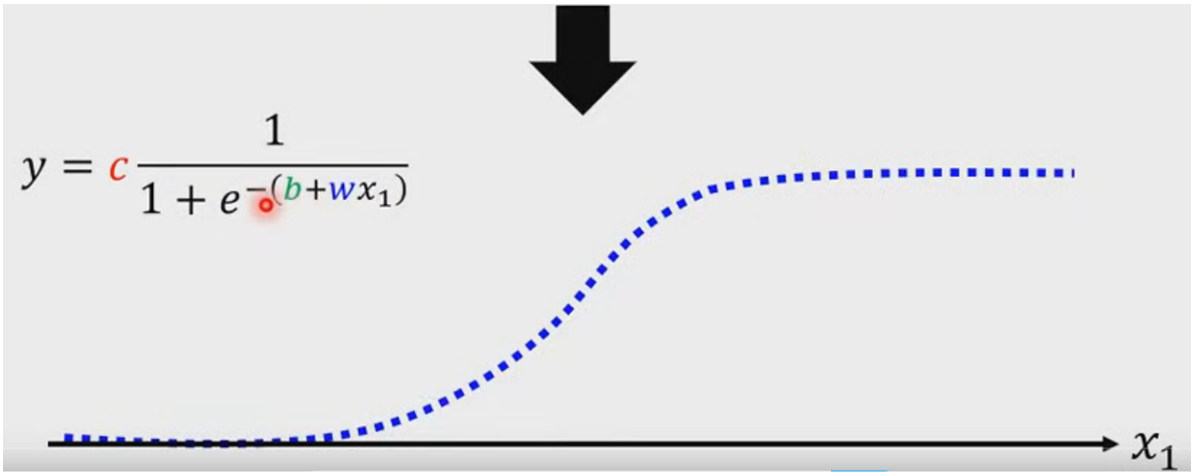
linear - more sophisticated models

model bias(模型本身限制)



用更好的模型(一组蓝线：分段线性)来近似。(有足够的分段线性，合成任意连续的曲线)

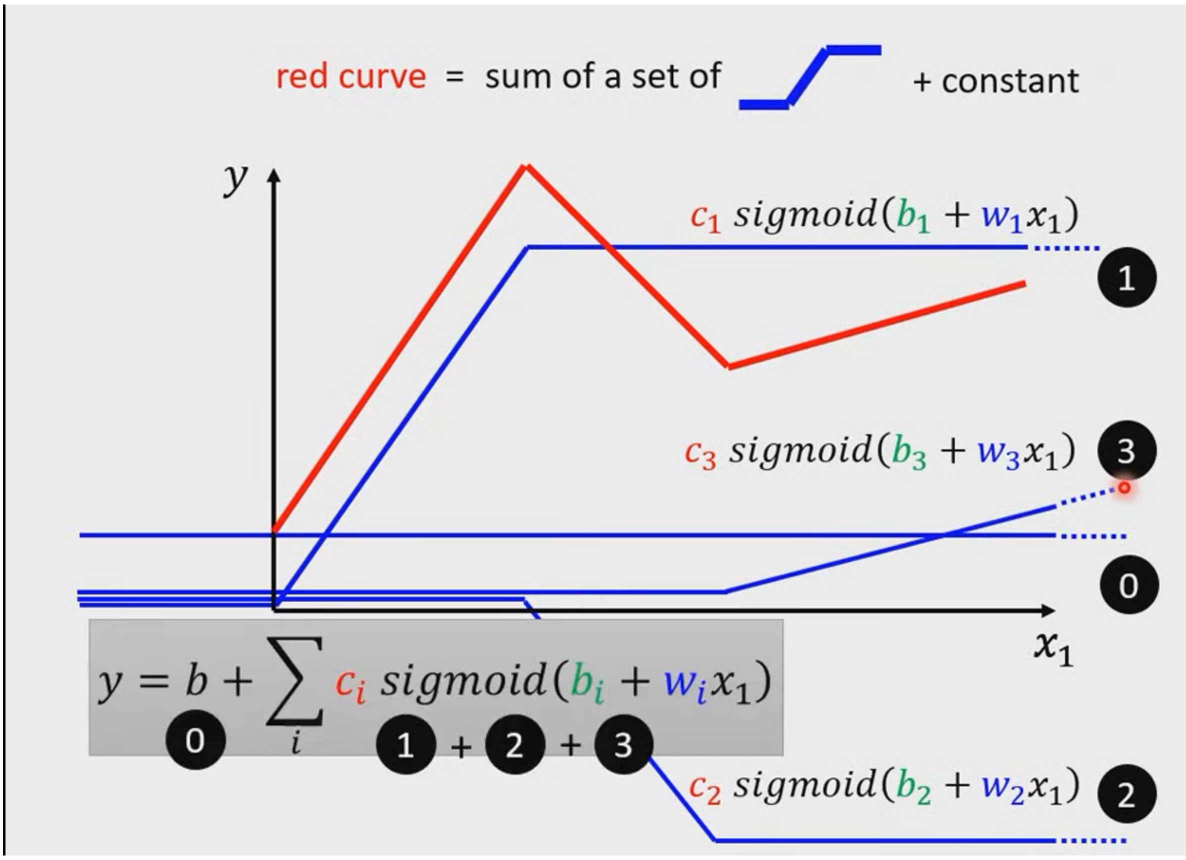
--sigmoid function



所以可以使用足够多的**sigmoid的线性组合**来拟合任意的连续曲线。

此前的分段线性蓝线称作“hard sigmoid”。

修改w,b,c得到不同形状的sigmoid，叠加得到连续的曲线。如下:

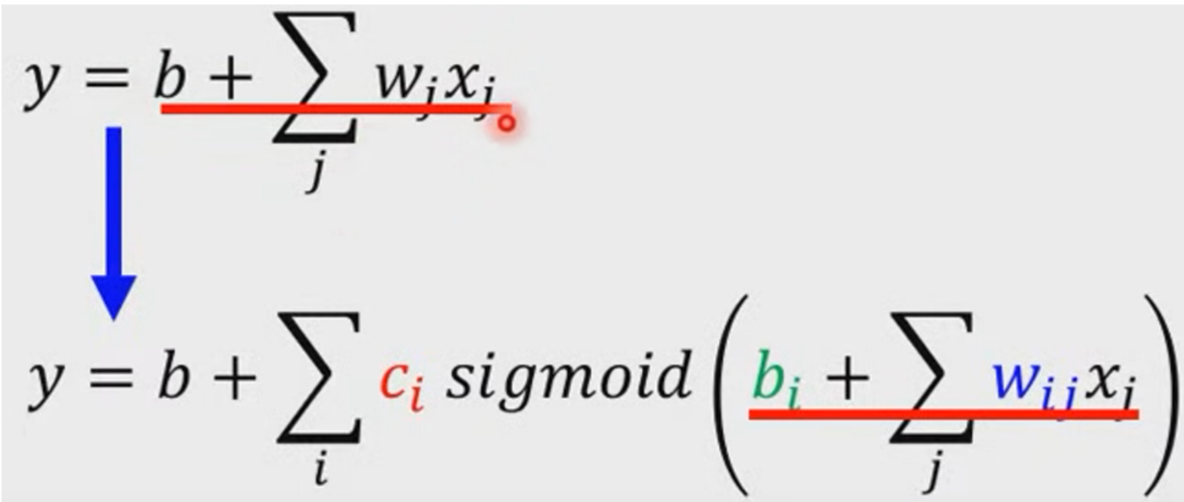


3个分段线性函数组合得到了所需拟合的分段线性函数。

通过设定不同的b,w,，叠加后逼近function:

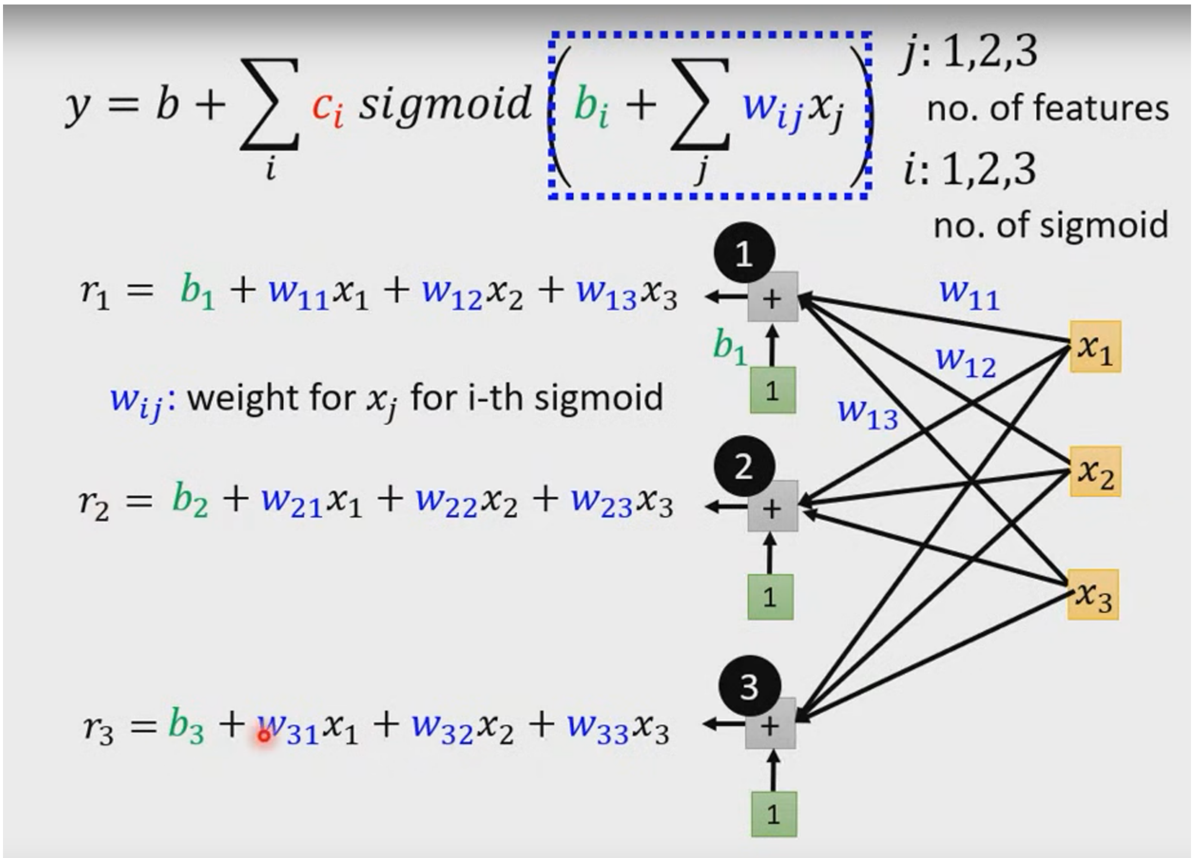
$$y = b + \sum_i c_i * sigmoid(b_i + w_i x_i)$$

将多个feature组合进行优化的操作可以由上述的“弹性”function得到:



这样能使得到的结果更有弹性。

对以上得到的公式作结构图，作图得到的结果如下:



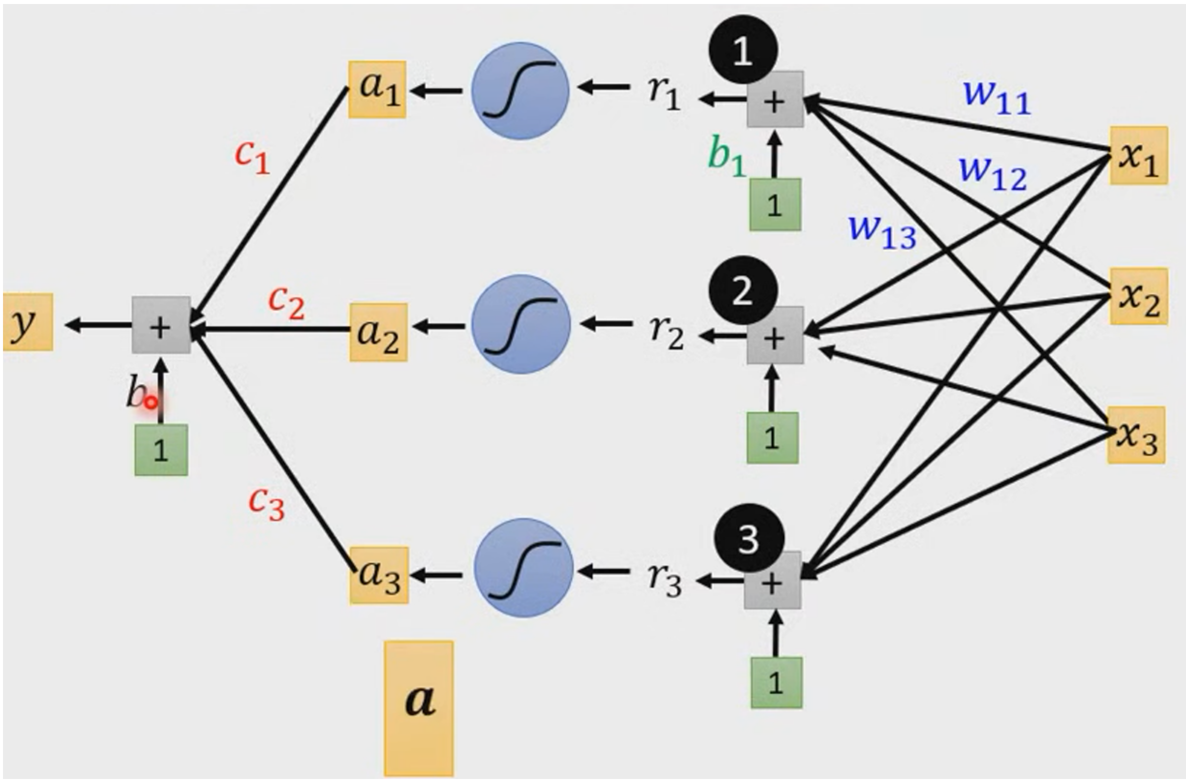
可以用一个矩阵描述:

$$r = Wx + b$$

最终的输出:

$$a = \sigma(r) = \sigma(Wx + b)$$

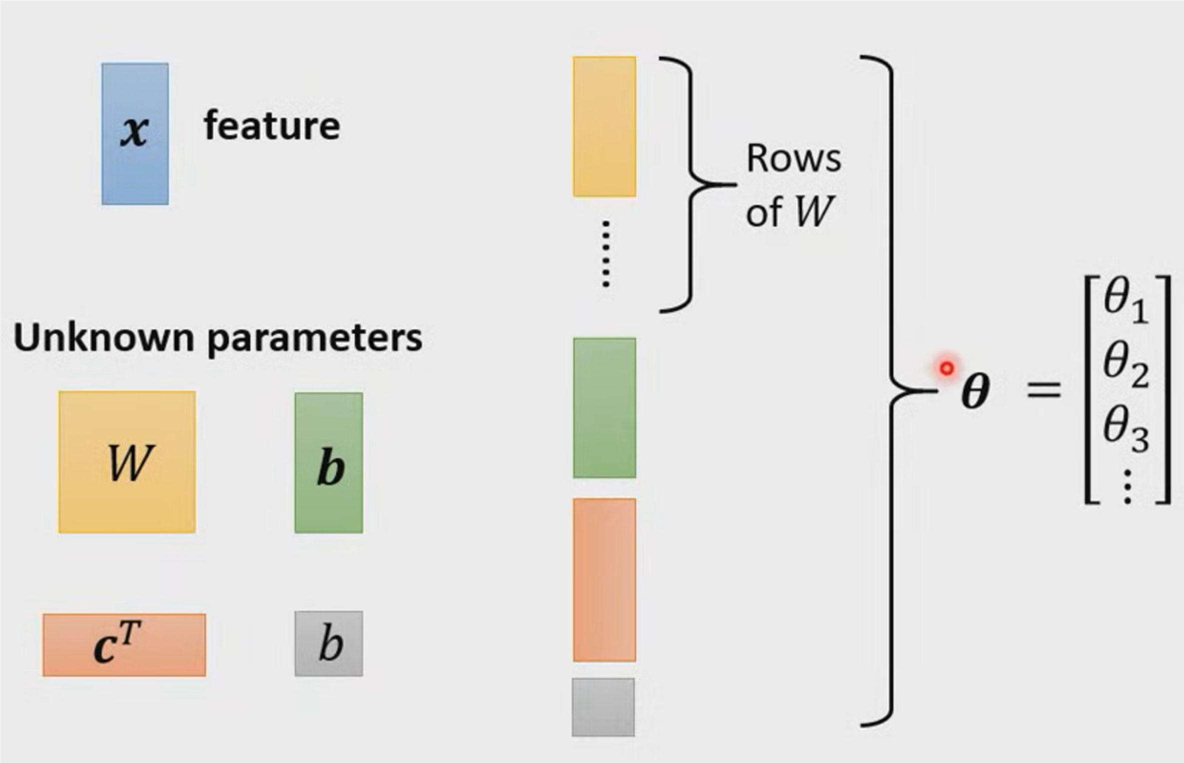
$$y = b + c^T a$$



综合:

$$y = b + c^T \cdot \sigma(b + Wx)$$

其中定义:



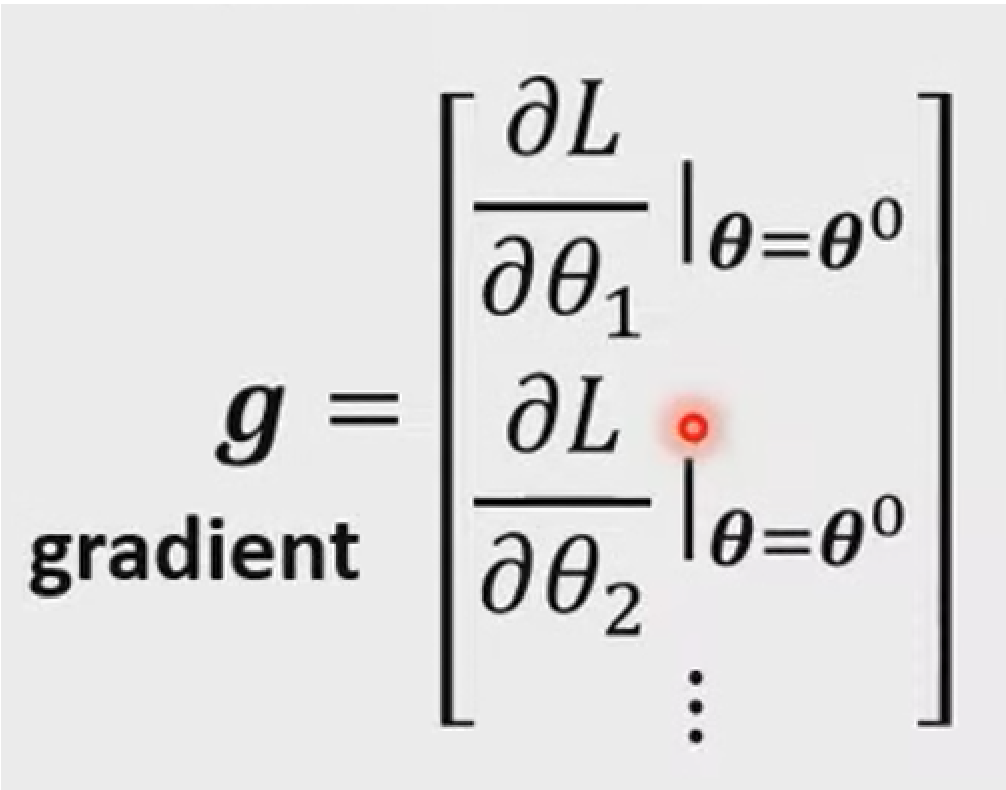
通过以上的形式改写了机器学习的第一步。

step 2： 定义Loss function

θ表示为一组参数向量，L对每一项作偏导并梯度下降。

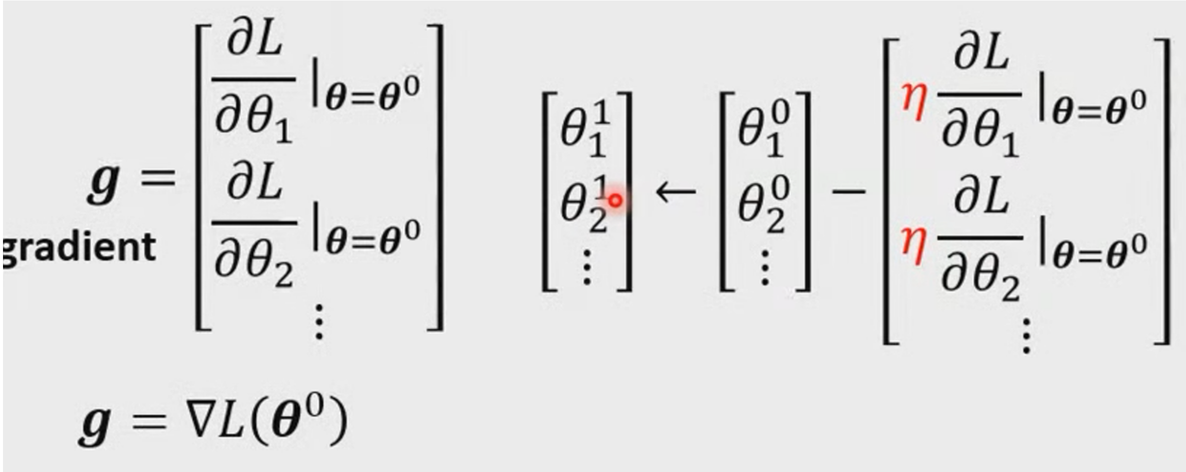
——gradient descent

g:所有的参数对L作微分得到的结果。



$$g = \begin{bmatrix} \frac{\partial L}{\partial \theta_1}\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \frac{\partial L}{\partial \theta_2}\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \vdots \end{bmatrix}$$

综合，对原参数作梯度下降。update方法如下:



也可简写作:

$$\theta_1 = \theta_0 - \alpha g$$

重复更新参数至结束。

Batch

将数据随机分成N组(n个batches)。

每次都对当前batch中的资料计算Loss, 用得到的gradient来更新参数，每次都用一个batch来更新参数。用所有的batch都对参数更新一次的操作记为一个**epoch**。

epoch : see all the batches once

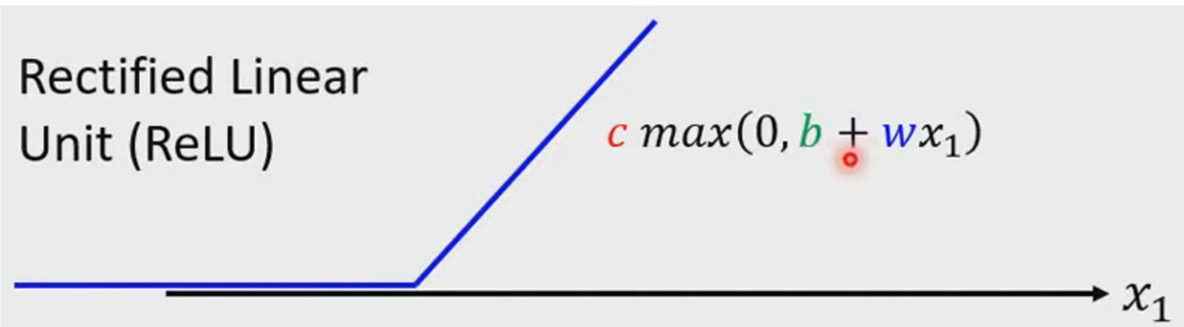
example :

10,000 examples & Batch size = 10 ---1000 batches 故每个epoch都对参数更新了**1000次**

batch size 也是一个超参数。

对模型作更多的变形

Relu :



输出 :

$$max(b + wx_1, 0)$$

将两个relu叠加可以得到一个hard sigmoid函数:

$$y = b + \sum_{i} c_i \, sigmoid\left(b_i + \sum_{j} w_{ij} x_j\right)$$

$$y = b + \sum_{2i} c_i \, max\left(0, b_i + \sum_{j} w_{ij} x_j\right)$$

哪一种较好?

Relu可能较好。

有很多层: Deep learning

网络越深未必越好--overfitting